NaSPA technical SUPPORT

Supporting Enterprise Networks and Operating Environments

JANUARY 1996
VOLUME 4, NUMBER 1

Introducing Access Control for VSE/ESA 2.1

Making the Transition to Parallel Sysplex at Boeing

Managing Implementation Projects for New and Emerging Technology

Using Process Configuration Management for Project Management

NetWare Resource Security: NDS vs. Bindery



NaSPA's Home Page
http://www.nascom.com/

technical®
Supporting Enterprise Networks and Operating Environments
SUPPORT

JANUARY 1996
VOLUME 4, NUMBER 1

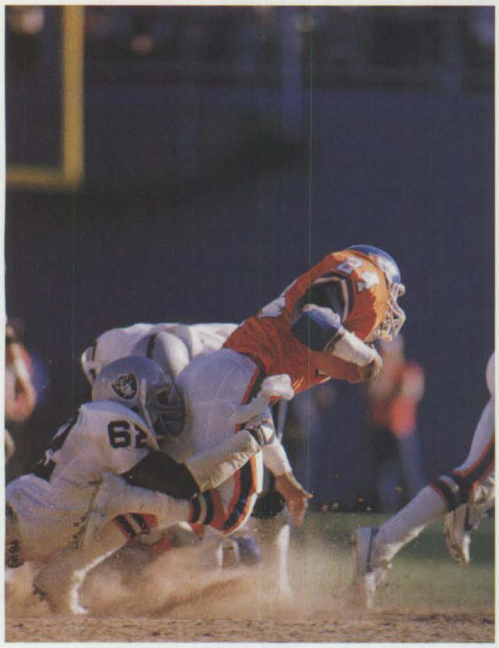


Photo: John Kelly, Image Bank, Chicago

FEATURES

10 Introducing Access Control for VSE/ESA 2.1
VSE/ESA 2.1 comes equipped with a substantial and sophisticated security feature. Here's an introductory look.
By Linda Dockstader

18 Making the Transition to Parallel Sysplex at Boeing: Part I — Motivation and Positioning
This article presents some reasons for making the transition to Parallel Sysplex and some steps required to position for it. Part II will discuss the new CMOS processors used in a Parallel Sysplex.
By Michael M. DuPas and Thomas Sible

22 Managing Implementation Projects for New and Emerging Technology
Useful tips for getting complex projects moving.
By Leo A. Wrobel

30 Using Process Configuration Management for Project Management
A configuration management system can simplify the change management process by tracking changes to code, documentation, requirements, etc., and linking these changes to report defects, project tasks, or function points.
By Sohail Haque

50 NetWare Resource Security: NDS vs. Bindery
This article examines major differences between NetWare 3.x bindery security and 4.1 NDS security. The information is designed to help current 3.x sites understand the benefits of going to NDS-based network security and management, and to help new 4.1 sites gain control and direction of NDS security.
By Guy C. Yost

NaSTEC '96
May 5th - 8th, 1996
Orlando, Florida

Celebrating our 10th Anniversary!
Come and join us by attending the best education conference yet for 1996!
Featuring keynote speaker, former President George Bush.

For more information, call NaSPA at (414) 768-8000, Ext. 109.

NaSPA Mission Statement:
The mission of NaSPA, Inc., a not-for-profit organization, shall be to serve as the means to enhance the status and promote the advancement of all corporate computing technical professionals; nurture member's technical and managerial knowledge and skills; improve member's professional careers through the sharing and dispersing of technical information; promote the profession as a whole; further the understanding of the profession and foster understanding and respect for individuals within it; develop and improve educational standards; and assist in the continuing development of ethical standards for practitioners in the industry.

The information and articles in this magazine have not been subjected to any formal testing by NaSPA, Inc. or Technical Enterprises, Inc. The implementation, use and/or selection of software, hardware, or procedures presented within this publication and the results obtained from such selection or implementation, is the responsibility of the reader.

Articles and information will be presented as technically correct as possible, to the best knowledge of the author and editors. If the reader intends to make use of any of the information presented in this publication, please verify and test any and all procedures selected. Technical inaccuracies may arise from printing errors, new developments in the industry and/or changes or enhancements to components, either hardware or software.

The opinions expressed by the authors who contribute to ***NaSPA Technical Support*** are their own and do not necessarily reflect the official policy of NaSPA, Inc. Articles may be submitted by members of NaSPA, Inc. The articles should be within the scope of host-based, distributed platforms, network communications and data base, and should be a subject of interest to the members and based on the author's experience. Please call or write for more information. Upon publication, all letters, stories and articles become the property of NaSPA, Inc. and may be distributed to, and used by, all of its members.

NaSPA, Inc. is a not-for-profit, independent corporation and is not owned in whole or in part by any manufacturer of software or hardware. All corporate computing professionals are welcome to join NaSPA, Inc. Membership rates are $39.95/year (USA), $49.95/year (Canada) and $59.95/year (all other countries). $19.98 of your annual dues is allocated to the publication ***NaSPA Technical Support*** and is non-deductible therefrom.

NaSPA Technical Support (ISSN 1079-3135) (IPM Agreement Number 0806773) is published monthly by Technical Enterprises Inc., 7044 S. 13th Street, Oak Creek, WI 53154-1429. Second-class postage paid at Oak Creek, WI and additional mailing office. POSTMASTER: Send address changes to ***NaSPA Technical Support***, 7044 S. 13th Street, Oak Creek, WI 53154-1429.

All product names mentioned in this publication are the trademarks/registered trademarks of their respective manufacturers.

INTEROPERABILITY INSIGHTS



25 MQSeries: Part I — What is Message Queuing?
A systems programmer describes the successful integration of messaging-enabled applications across major platforms.
By Tom Kruczek

42 Routers and Firewalls: Packet Filtering
This article examines the headers in TCP/IP and Ethernet II network layers, and shows the filtering possibilities.
By Mark Bell

53 The Law and Your Plans for Doing Business on the Internet
This article discusses how keys and certification authority can help make electronic commerce more secure.
By Charlotte Ann Griffin

55 Profiting Through Effective Network Management: Part III
The network of tomorrow will evolve into a "centralized client/server" to provide greater security and standardization, reduced equipment cost, and reduced personnel cost at the customer location.
By Leo A. Wrobel

58 Getting the Most From Your Presence in Cyberspace
A Web presence can help your company promote its products, and provide timely customer service and support. But, it involves more than just getting on the Web with a home page.
By G.A. "Andy" Marken

SYSTEM STRATEGIES



13 Building an ISPF-Based Corporate Management Information System: Part II
The first part of this series described a corporate management information system running under ISPF. This article describes implementation procedures.
By Michael Swetlow

34 Data Compression: Reducing Transmission Costs Without Sacrificing Data
Implementing compression software can help companies reduce data transmission time and costs. This article describes lossy and lossless compression techniques, and the cross-compatibility aspects of PKZIP.
By Shannon Wendel

47 Using Programs as Functions in OS/2: How to Use DosStartSession
You can use the DosStartSession application programming interface (API) function to invoke a non-OS/2 program.
By Bruce Eric Högman

COLUMNS

60 MVS Tools & Tricks
Keeping Up With the Unit Control Block: Part II
By Sam Golob

63 VSE Tools & Techniques
A Subroutine to Obtain the POWER Job Number of a Currently Executing Job
By Mark Hanna

64 VM Toolbox
Animated PIPELINES
By John D. Kinne

66 Storage Strategies
Too Much of a Good Thing — Multivolume Data Sets
By Steve Pryor

68 OS/2 Insights
The Case of the Dead Gateway
John E. Johnston

69 NetWare News
Security Using Gateways
By Guy C. Yost

71 Opening Windows
"Dream Trip" to Asia Includes Sightseeing of Gadgetry
By Al Shing

74 Security Strategies
Staving Off the Would-Be Hacker
By Eric Allred

76 On a Personal Note
The Customer Ain't Always Right
By Mike Sutton

DEPARTMENTS

6 From the President

8 NaSPA News

37 NaSTEC '96

77 Product Profiles

BY BRUCE ERIC HÖGMAN

Using Programs as Functions in OS/2: How to Use DosStartSession



You are developing an OS/2 application and need to do a function that you find is done by an existing program. That program may not be an OS/2 program. No problem. You can use the DosStartSession application programming interface (API) function to invoke that program as a subprogram or external function of your own.

You can use the DosStartSession application programming interface (API) function to invoke non-OS/2 programs.

Why use an existing program to perform a function? Here are several reasons:

- An existing program runs successfully.
- The interface to an existing program is well-defined and isolated from your own application. It fits into your modular approach.
- You do not want to take the time to reproduce the existing program functions in your own code, even if you have the source code to that existing program.

Sometimes using an existing program is an interim measure to support critical processes while you are creating and putting in place another permanent application solution. Whatever the reasons, you need to know how to invoke external programs as functions of your own. This article addresses the several alternative API function calls and shows you how to use DosStartSession as a general purpose solution.

You can view information on DosExecPgm syntax, data structures such as STARTDATA, and return codes for all API functions in the softcopy documentation included with the Developer's Toolkit and with VisualAge C++. That information is not printed here due to space. The RXRUN package, containing C source code for a C function and for a REXX external function, is available in the OS/2 LIB of NaSCOM as filename RXRUN.ZIP. The source contains examples of calls to the API functions discussed in this article, other than DosExecPgm.

API FUNCTIONS

You can use several API functions to invoke other programs. These are described in general. Detailed information on DosExecPgm and DosStartSession is contained in "Control Program Reference." WinStartApp is described in "PM Reference."

DosExecPgm is a control program API you can call in your program to invoke a program of the same session type. This is a "low level" function. You can invoke a program as a child process and have your program wait as a parent process on the termination of the child, or have the child program run independently. The syntax for DosExecPgm is shown in Figure 1.

A session consists of one or more processes. Each has one or more threads. A session has a virtual console or screen that is windowed, full-screen, or has a Presentation Manager (PM) window, together with buffers for keyboard and mouse input. Where DosExecPgm creates child processes, DosStartSession creates child sessions.

Using only DosExecPgm API limits what programs your OS/2 application can launch. If you are developing a PM application that will call only other PM application programs to use as functions of your application, you should use DosExecPgm to establish a parent process and child process relationship. This will take advantage of some features available only to processes in a parent and child relationship, such as DosSetPriority and DosKillProcess. Those considerations are beyond the scope of this article.

WinStartApp is a Presentation Manager (PM) API that uses workplace shell services in starting program sessions. You can invoke programs of all types using this API function. WinStartApp calls DosStartSession to create a new session after it processes arguments passed to it.

DosStartSession gives you all of the features you need to start program sessions of any type. This article presents technical details on how to use the DosStartSession API.

Figure 1: DosExecPgm

```
#define INCL_DOSPROCESS
#include <os2.h>

PCHAR      pObjname;
   /* points to the buffer holding the name of the object related to
      the failure of DosExecPgm  */
LONG       cbObjname;
   /*  Length in bytes of the buffer.  */
ULONG      execFlag;
   /*  Flag saying if program is child process and if debugging is
      enabled.  */
PSZ        pArg;
   /*  points to argument strings  */
PSZ        pEnv;
   /*  points to environment strings  */
PRESULTCODES pRes;
   /*  points to RESULTCODES structure  */
PSZ        pName;
   /*  points to name of executable  */
APIRET     ulrc;
   /*  return code  */

ulrc = DosExecPgm(pObjname,
        cbObjname, execFlag, pArg,
        pEnv, pRes, pName);
```

DOSSTARTSESSION

The following sections refer to data contained in the online softcopy. Code samples use **typedef** names defined in the header files. Refer to the .h header files in the toolkit or in the C++ INCLUDE directories to obtain the current data structure definitions. **Note:** Discussion of "program handles" in "Control Program Reference" related to API functions WinAddProgram and WinQueryProgramHandle refers to OS/2 V1.x functions no longer documented. Later versions of the reference may omit these functions.

Details of tailoring the window session are passed to the operating system using the STARTDATA structure. In this structure, the Related variable indicates whether the new session will be a child session or an independent one. The SessionType indicates the type of window session: PM, windowed, or full-screen.

Your understanding of DosStartSession may be complicated by all the information you provide in the STARTDATA structure. However, much of the information can be left NULL or zeroes, and the operating system will provide safe default values.

The checklist for filling in the STARTDATA information and coding the call to DosStartSession is fairly simple if you think of it as analogous to filling in the entry fields of a settings notebook for a WPProgram object:

- Name the executable file to invoke. The drive and path may not be required if the program is in the PATH of the current environment.
- Give the parameters to pass to that program.
- Indicate what type of session (PM, OS/2 window, OS/2 full-screen, etc.).
- Indicate whether the child session will be related or independent.

Some changes are required for WINOS2 programs. To launch a WINOS2 program, follow the checklist outlined below:

- PgmName should be: c:\os2\mdos\winos2\winos2.com (or the boot drive for your system instead of C:)
- PgmInputs should be: /3 winappname winappopts

Here, "/3" tells WINOS2 to run in enhanced mode. Other values are default mode (no parm), real mode (/r), and standard protected mode (/s). Default mode is set by the WINOS2 Setup object in the OS/2 System Setup folder.

"winappname" is the executable filespec that is the target WINOS2 application program you want to invoke.

"winappopts" are the parameters to pass to the target WINOS2 application program.

- Set SessionType to SSF_TYPE_VDM for full-screen session or SSF_TYPE_WINDOWEDVDM for seamless WINOS2.
- Set Related to the value you want.

The PgmName and PgmInputs strings combined length may not exceed 1,024 characters.

You can start a command prompt window session by setting the PgmName address pointer to zero or by providing a null string and setting PgmHandle to zero. The OS/2 program specified in the SET OS2_SHELL statement in CONFIG.SYS for OS/2 session types or in SHELL= for DOS session types is executed and uses PgmInputs string as its arguments.

Before calling DosStartSession, call DosCreateQueue to create a queue used as a termination queue to receive notification messages about tie termination of the child session. You do not need to do this if you will create an independent session. See "Creating a Termination Queue."

If you do not know the type of program you will invoke, you can let the operating system decide the session type based on an interpretation of the target load module. It is better, in general, to know the type of program you want to use as a subprogram and to set up the call to DosStartSession appropriately.

CHILD OR ORPHAN?

You can start a child session and wait on its completion or create it as an orphan — an independent, unrelated session. You can also indicate that you want to be notified upon its completion or discard the termination information.

A related session is called a child session. The parent session starts the child session. Sessions started as unrelated are independent sessions, not child sessions. An application can control its child sessions by using the identifier returned by DosStartSession with the DosSetSession, DosSelectSession, and DosStopSession. Only child sessions can be controlled. The Related field in the STARTDATA structure specifies whether the new session is related to the session calling DosStartSession. A session ID is returned only for child sessions. When a parent session terminates, then all of its child sessions are ended.

A child session may itself call DosStartSession to become a parent session. There is no relationship between a parent session and its "grandchildren." However, a child session terminates when its parent does. This holds true for each session in a "family tree".

FOREGROUND OR BACKGROUND?

New sessions may be started in the foreground (i.e., have desktop window focus) only when the caller's session (or one of the caller's descendant sessions) is currently executing in the foreground. The foreground session for windowed applications is the session of the application that owns the window focus. The new session appears in the Shell switch list, and it is given window focus. An attempt to start a foreground session from a session not in the foreground will return a code 457 from DosStartSession. This indicates that the new session was started in the background, and the new session is started otherwise successfully. Your own program logic should anticipate this and accept both zero and 457 as successful return codes.

A parent session can call DosSetSession to tell the workplace shell to set focus on a child session when the parent session is selected through the desktop window list. This is called establishing a bond. Only one child session at a time may be in a bond with its parent.

Use STARTDATA PgmControl to specify whether you want the new session to start minimized, maximized, and whether a WINDOWABLEVIO session will have AUTOCLOSE in effect (the default). Sessions normally start not minimized or maximized.

DYNAMIC LIBPATH

When creating child sessions, remember that the child inherits the current environment your application has, unless you change the environment strings, current directory, and other system information. Under OS/2 Warp V3, you can also extend the LIBPATH using SET BEGINLIBPATH= or SET ENDLIBPATH= for environment variables to specify that path to prepend or to append to the LIBPATH specified in CONFIG.SYS. This lets you dynamically add .DLL file search paths to the environment for the current window session and its child sessions. Remember that the current directory on each hard drive is part of the environment and affects relative path values (those that do not start with '\').

Technical developers are eager to exploit operating system services to make their applications run as fast as possible. They tend to use API functions that support tightly coupled communication. These include shared memory, pipes, queues, and semaphores. In linking separate application programs to perform complex functions, I advise developers to use the highest level and most dependable method of communicating data that fulfills the functional requirements. Most often, this results in use of files, either permanent or temporary, to pass data among functional modules.

Use of files for passing data is supported on all platforms and across all networks. True, it is often not as fast as more closely coupled methods of data communication, but it is often more reliable and less prone to timing problems, memory "leaks," abnormal termination of processes, etc. You can also develop your own multiple program systems using files initially and then change to other methods once the bugs are fixed. Recording messages and data in files usually leaves excellent debugging footprints to help locate problem areas.

CREATING A TERMINATION QUEUE

Call DosCreateQueue to create a termination queue. Use the HQUEUE handle returned as the STARTDATA TermQ variable in calls to DosStartSession and DosReadQueue.

Queue names follow rules for OS/2 file names (although no physical file is created). The queue name is prefixed by '\QUEUES\' followed by a unique file name [and extension]. An example: \QUEUES\myqueue.que.

The process that creates a queue is the server process of that queue. Other processes that access the queue are client processes.

CHILD SESSION TERMINATION

Call DosReadQueue with a zero value for the wait parameter to detect when a child session has ended. The REQUESTDATA variables pointed to will contain the PID of the process of the child session and the result code. The data element structure is two unsigned long integers:

```
ULONG  PID of child
ULONG  Result code
```

Call DosFreeMem to free the segment containing the data element, after processing the data element.

Your application may also use the queue created as a termination queue for other communication between sessions if you pass a unique request identifier via DosWriteQueue. Request identifier values zero (0) through 99 are reserved by the operating system, so use values of 100 or greater.

The result code returned in the TermQ data element is that of the program named in the PgmName variable. When a secondary command processor (CMD.EXE) is named and the target application is named in the parameters passed to the command processor, the result code is that of the target application program. This is provided that the command processor parameters included /C to close the command processor session immediately upon termination of the application program. If you use /K instead of /C when you invoke a command processor, then the result code will be that of the command processor itself.

The parent session becomes the foreground session when a child session terminates while having focus, i.e., when the child session is the foreground session. When independent sessions terminate, the workplace shell selects the next foreground session.

When you use DosStopSession to terminate a child session, a process running in that session can ignore the request to terminate. DosStopSession still returns zero result code, indicating success. Only by waiting for notification by using DosReadQueue to read the termination queue can your parent session know when the child session has terminated.

Using DosStopSession breaks any bond established between the parent session and that child session by DosSetSession.

DEBUGGER CONSIDERATIONS

Consider using TraceOpt value SSF_TRACEOPT_TRACEALL (2) to trace all descendant sessions when you want to run debuggers. This requires you also create child sessions and have a termination queue. The debugger must also call DosReadQueue to get notification when a child session is created and free segments containing queue data elements it removes using DosFreeMem.

RESTRICTIONS

You cannot call DosStartSession from a detached session. A detached session is one that was started using DETACH on the command line or by a call to DosExecPgm using the EXEC_BACKGROUND flag.

You cannot use API functions that require a process relationship between the parent session and child session that is a parent process and child process, since DosStartSession does not establish a parent process and child process relationship. The parent process of the child session is the shell process. The ppid (pointer to the PID) returned by DosStartSession may not be used in calls to system API functions, such as DosSetPriority or DosWaitChild, which require a parent process and child process relationship. Only DosExecPgm can establish a parent process and child process relationship.

Despite these minimal restrictions, you should find DosStartSession a worthwhile addition to your programming notebook. Why reinvent a function when you can simply invoke an existing program? The RXRUN package, available on NaSCOM, contains C source for functions callable from C or REXX programs to do the DosStartSession function. You can use the source as examples of coding your own programs to use DosStartSession, or call the functions already packaged. ts

REFERENCES

The following references are distributed as documentation softcopy .INF files with the IBM VisualAge C++ product 5622-017 and with the Developer's Toolkit for OS/2 Warp.

"Control Program Reference": IBM OS/2 Control Program Guide and Reference, provides technical information on APIs that call control program API services.

"PM Reference": IBM OS/2 Presentation Manager Programming Guide and Reference, provides technical information on Presentation Manager API services.

Was this article of value to you? If so, please circle Reader Response Card No. 38.

Bruce Eric Högman is employed by Electronic Data Systems Corporation, in its South Florida Resource Center, Boca Raton, Fla. He is a consultant to IBM in Customer Support, Migration Support. Bruce has been in data processing since 1965. He has been an instructor at the USMC Computer Sciences School, Quantico, Va., where he taught IBM 360 Assembler Language to Marines. He may be reached at 72050,1327@compuserve.com.

The Case of the Dead Gateway

BY JOHN E. JOHNSTON

Anytime you get a call at 3 a.m., you know it's not going to be good news. The conversation went something like this:

COMPUTER OPERATOR: "Hey John. Sorry to wake you, but we're have some weird problems."
ME: "Tell me about it."
OPERATOR: "None of the OS/2 workstations can access the mainframe."
ME: "What kind of errors do they get?
OPERATOR: "It says COM695 at the bottom of the emulator screen."
ME: "How about the DOS and Windows workstations? Can they access the mainframe?"
OPERATOR: "Yes."

We have a 3745 front-end processor in our shop. Each OS/2 workstation has Communications Manager/2 installed. These OS/2-CM/2 workstations then go through one of three dedicated CM/2 gateways to access the mainframe. We have 10 Ethernet segments that run from the computer room to the various floors and buildings in the organization. These segments are connected using a high end router. We also have a small Token-Ring network which is isolated in the computer room.

The three CM/2 gateways each have one Ethernet network adapter and one Token-Ring network adapter. The Token-Ring connection is used to connect the gateway to the 3745. The Ethernet connection is used to connect the gateway with the rest of the network. The translation of the mainframe traffic from Token-Ring to Ethernet is performed in the gateway. To enhance availability, each gateway is connected to a different Ethernet segment.

Each DOS and Windows workstation has Rumba loaded as the mainframe terminal emulator. These workstations then access the mainframe via one of two NetWare for SAA gateways. The SAA gateways are connected to the 3745 via a Token-ring connection.

ME: "Are the CM/2 gateways up?"
OPERATOR: "Yes. We also tried recycling them."
ME: "Can the OS/2 workstation access any LAN applications like WordPerfect or Lotus?"
OPERATOR: "Yes, everything works except the mainframe connection."

The offending connection was simply an un-terminated wire sticking out of the wall plate.

I was perplexed. What could affect the OS/2 workstations yet leave the DOS and Windows workstations alone? It couldn't be the 3745 or the SAA gateways would also be affected. It couldn't be a problem with the Token-Ring network or the DOS users would also be affected. If it was a router problem, the users wouldn't be able to access any LAN applications. I doubted if all three CM/2 gateways could be affected at the same time . . . or could they?

I made a trip to the office. When I arrived, I checked the CM/2 gateways and they looked fine. I then went to my office where I have both an OS/2 workstation and a Windows workstation and saw the problem. "This makes no sense," I said in disbelief.

Then I started turning things off. My plan was to turn off all of the non-critical LAN components to isolate the problem. I turned off all of the print servers, remote access gateways, SAA gateways, and all but one of the CM/2 gateways. At this point, the only two devices active on the Token-Ring network were the 3745 and the single CM/2 gateway. No luck. The problem was still there. Next, I recycled the 3745. Same story . . . no host access.

Now I was getting worried since I didn't know what could be causing this problem. I contemplated emptying the bank account and moving to the Virgin Islands, but reality set back in.

I then went to the network monitor which is an SNMP-based package. It shows, among other things, the bandwidth utilization of each network segment. I was methodically looking at each Ethernet segment and each was in the 10 to 20 percent range, which is normal in this environment. Then I noticed that one of the segments was up to 50 percent. With this particular monitor, I could drill down to see what workstation was causing the bulk of the traffic. The majority of the traffic was caused by a single workstation in a new office that had just been wired. "I didn't know that workstation was hooked up yet," I thought to myself. "Surely no one would install a workstation without letting me know."

I walked up to the offending workstation and I couldn't believe what I saw. The area where the new wire had been run utilized a small Synoptics concentrator. Twisted-pair wiring was run between the concentrator and the workstation to provide LAN connectivity. The offending connection was simply an un-terminated wire sticking out of the wall plate. The ends of the wires were stripped of insulation and twisted together. I untwisted the wire, then went back to my office to see if this fixed the problem. The host access was operational again.

I still don't know why this affected only the CM/2 host access, but it was a lesson well learned. Keep your fingers crossed, not your wires!

If you have any ideas for future topics for this column, please contact me at 73473.2146@compuserve.com. ts

Was this column of value to you? If so, please circle Reader Response Card No. 46.



NaSPA member John E. Johnston is manager of technical support and communications for a major hospital in Pennsylvania. He designs and maintains cross-platform local and wide area networks utilizing NetWare, OS/2, DOS, and Windows.